Interface subroutines for IPCRESS files

R. E. Clark, J. Abdallah, Jr., and H. L. Zhang

Abstract

In this document we describe a set of subroutines recently developed to access both gray and multi-group opacity data stored on files with the new IPCRESS format. These subroutines allow the user to Get All Needed Data from an Opacity Laden File (Gandolf). We describe each subroutine available to the user with examples of calling the various subroutines. In an appendix we list all the subroutine names and the comment sections for the user callable subroutines. We also give a description of the IPCRESS file in an appendix.

I. Introduction

Over the years there have been a variety of file formats used for the storage of opacity data. Some files were sequential binary files, such as the MIXUP format; others were random access binary files, such as STYX and PARADISE. These formats were developed on the Cray computer platforms. There are some problems with all existing formats, such as mixture of integer and floating points numbers within a record, mixture of Hollerith with numerical data, and packing of two floating point numbers in one word. These difficulties make it necessary to define a new file format that is more transportable across machine boundaries. However, it is desirable to retain the random access nature of the files for quick access of an arbitrary record within a large file. We have thus modified one of the random access file formats, the PARADISE format, to make it Independent of Platform and Can be Read by existing Software Subroutines (IPCRESS files). In this document we describe the various Gandolf subroutines for accessing data from an IPCRESS file and describe test code using these subroutines along with some sample opacity files in the IPCRESS format.

A full list of all names of subprograms used in the Gandolf Load library is included in Appendix A. A listing of the comment sections of each user callable subroutine, including information on error flags for each subroutine, is available as Appendix B. A list of the keywords for opacity related data appears in Appendix C. A description of the IPCRESS file format is included in Appendix D.

In the next section we describe subroutines for retrieving general information from a file. In section III we describe subroutines for reading in any data record from a file. Section IV describes some subroutines for interpolating opacity data. The final section describes the method for accessing the files containing the load libraries and test code along with sample opacity files for various computing platforms.

II. Subroutines for general information

In this section we describe the Gandolf subroutines for finding general information from an opacity file. This includes obtaining a list of materials on a file, a list of types of information for a given material, and seeing the sizes of the temperature, density, and, if applicable, the photon energy grids for a given material on a given file.

We note here that the Gandolf subroutines do not do any memory managing. It is up to the host code to set up arrays to hold any data to be read in. The Gandolf subroutines will check the amount of data found on a file against the amount of space the user has set aside. If there is not enough space, the Gandolf routines will not perform the read. Using the Gandolf routine for checking sizes of grids is recommended before reading the data records.

We also make a note about file handling. The Gandolf subroutines open a file only when necessary to perform a read. When the read is finished, the file is closed, so there are no files remaining open upon return from the Gandolf subroutines.

Finally, we make a note about integers. On the Cray platforms all variables were eight bytes in length. On other platforms this is not necessarily the case. On most platforms it is necessary to specify the length of both floating points and integers. We have chosen to compile the Gandolf subroutines with eight bytes for both floating points and integer variables. The data on the IPCRESS files is either floating point of length eight bytes, or character; there are no integers on the files. If a user has a need to have integers passed into and out of the Gandolf subroutines as four byte integers, the code can easily be recompiled. Please contact data_team@lanl.gov (or phone 665-7676) for more information.

Finding materials on a file

Let us assume the user has an IPCRESS with name fname available for reading. The variable fname is character*80 and may contain a path name as well as file name. Valid examples would be

```
testfile
/usr/tmp/rehc/testfile
```

where in the first case the file testfile needs to be in the local area where the code is running. The user must set up an integer array to hold material identification numbers. We follow the same numbering convention as the Sesame library: All opacity material identification numbers (matid) are 5 digit numbers. The first digit is the numeral 1, which is the Sesame convention for opacity. The next three digits identify a material, and the final digit is a version number. As an example the following matids all refer to aluminum:

13710 13711 13717

in which one sees that 371 is the identifier for aluminum and there are versions 0, 1, and 7 listed. Of course, the user is not required to use the Sesame identifier for an element, nor is the user required to follow any version convention. However, the opacity table generating code, TOPS, will not permit matids outside the range 10000-19999. Also, the TOPS code will not permit the user to use the same matid more than once on an IPCRESS file. Thus the IPCRESS file will have a set of materials with matids in the range 10000-19999.

Let us assume that the user has defined an array called matids with dimension kmat:

```
dimension matids (kmat)
```

and then wishes to see a list of matids on the file named testfile. The following call will put the available matids in the array:

```
character*80 fname
fname='testfile'
call gmatids (fname,matids,kmats,nmats,ier)
```

where nmats is the number of matids found by Gandolf and ier is an error flag; see appendix B for values returned for ier. If ier is zero, then there will be nmats integers in the matids array, each representing the material identification number of a material on the file.

Finding keyword for a material

Once the array of matids has been obtained it is possible to find a list of all types of data available for that material on the specified file. The types of data are specified by keywords which are of type character*24. Each type of data has a unique keyword associated with it. Examples are 'rgray' for Rosseland gray opacities, 'pgray' for Planck gray opacities, 'tgrid' for temperature grid, etc. See appendix C for a full list of keywords and associated data types.

Assuming that the user has a file name set up and has read in the array of matids with the gmatids subroutine, the following call will read in all the keywords for a given matid:

```
Character*80 fname
character*24 keys (kkeys)
.
.
.
mat=matids(i)
call gkeys(fname,mat,keys,kkeys,nkeys,ier)
```

where i kkeys is the size of the array and nkeys is the returned value of the number of keywords found. The array keys contains nkey entries. If the returned value of ier is not zero, see Appendix B for interpretation of the value.

Finding sizes of grids

In order for a code using memory managing to set aside a proper amount of space for various arrays, the size of the temperature, density, and, if needed, photon grids must be known. This information is obtained with the Gandolf subroutine gchgrids. Assuming that the file name is assigned as above, and a variable mat is set to the desired matid, the call

```
call gchgrids(fname,mat,nt,nrho,nhnu,ngray,nmg,ier)
```

will provide the needed grid size information. In this example nt is the number of temperatures, nrho is the number of densities, nhnu is the number of photon group boundaries, ngray is the number of gray opacity points, and nmg is the number of multi-group opacities. The Gandolf code checks for consistency among these quantities and sets the error flag ier if a discrepancy is found. In other words, the relationships

```
ngray = nt*nrho
nmg = (nhnu-l)*nt*nrho
```

should be true, where nhnu-1 is the number of photon groups for multi-group opacities. With these grid sizes in hand, a memory manager can allocate all needed arrays for gray and multi-group opacities.

III. Reading in data records

There are three basic types of methods for retrieving data from the IPCRESS files using the Gandolf subroutines. First, the gray data can be retrieved along with the temperature and density grids. Secondly, multi-group opacities along with the temperature, density, and photon boundary energies can be read. Finally, a single data type can be read without any associated grids.

Reading gray data

For the purposes of this document we define gray data as any data that depends only on temperature and density. Thus Rosseland gray opacities is one example of gray data. Another example is the mean ion charge, which depends only on temperature and density, although it is not an opacity per se.

If we assume that a file has been defined as before, a matid has been selected and called mat, and space has been set aside for grids and gray data, the following will retrieve Rosseland gray opacities:

```
      Character*80 fname
      Character*24 key
      dimension temps(kt),rhos(krho),ross(kgray)
      .
      .
      .
      mat=matids(i)
      key=' rgray'
      call ggeLgray(fname,mat,key,temps,kt,nt,rhos,krho,nrho,
&   ross,kgray,ngray,ier)
```

where the Rosseland opacities will be put into the array ross. There will be ngray of these data if ier is zero. Otherwise, use Appendix B for the meaning or ier. Also, the associated temperature grid will be put in the array temps and the density grid will be put into the array rhos. All quantities are the actually data, not logarithms. The units for temperature are keV, densities are in gm cm^{-3} and opacities are in cm^2/gm. The gray quantities are stored with the following arrangement: There is one large array with all data contiguous. There are nt*nrho data points. The first nrho points are the data points for all densities for the first temperature. The next nrho points are the data points for all densities for the second temperature etc. This arrangement holds for all gray quantities.

Reading multi-group data

The reading of multi-group data is similar to reading the gray data with the addition of a third grid, the photon group energy. Note that if there are nhnu photon boundaries there will be nhnu-1 groups. The photon grid stored on the file has nhnu numbers; there are nhnu-1 groups.

The arrangement of data is similar to that of the gray, with the photon dependency being the innermost variable. Thus, the first nhnu-1 data points are the multi-group opacities for the first temperature and first density, the next nhnu-1 points are for the first temperature and second density etc.

As an example, to read in all the Rosseland weighted multi-group opacities for absorption the following code could be used:

```
      Character*80 fname
      Character*24 key
      dimension temps(kt),rhos(krho),hnus(khnu),ramg(kmg)
      .
      .
      .
      mat = matids(i)
      key = 'ramg'
      call ggetmg(fname,mat,key,temps,kt,nt,rhos,krho,nrho,
&   hnus,khnu,nhnu,ramg,kmg,nmg,ier)
```

where the multi-group data will be put into the ramg array provided that ier is returned as zero. For other values of ier refer to Appendix B. The units are as for the gray opacities with the addition of the photon energies, which are in keV.

Reading single data entries

At times it may be useful to read a data entry directly without referring to whether it is gray or multi-group data. For example, if it was desired to simply see the photon group boundary array, it would be desirable to be able to read in just that data without having to read in a full multi-group opacity array.

The reading in of a single data array can be accomplished by the following sample code:

```
Character*80 fname
Character*24 key
dimension data(kdata)
.
.
.
mat=matids(i)
key='hnugrid'
call ggetdata(fname,mat,key,data,kdata,ndata,ier)
```

which puts ndata values into the array data, provided ier is returned as a zero. For other values of ier refer to Appendix B. In this example the data read will be the photon energy boundary values.

IV. Interpolation subroutines

Since the opacities are on a fixed set of grids, it is necessary to interpolate on grid values for both gray and multigroup opacities. We recommend that the interpolation be a linear on log method. Also, for values that go off a table boundary we recommend using the value at the table boundary; no attempt at extrapolation should be made. The Gandolf system contains subroutines for this method for both gray and multi-group opacities.

We recommend that upon reading in opacity data that the log of all data be stored. This greatly speeds up the interpolation process. The two subroutines gintgrlog and gintmglog assume that all input quantities are log values. These routines return actual values for opacities on output. As an alternative, the subroutines gintgray and gintmg assume that input quantities are actual values, not logs. Output values are actual values, not logs.

To interpolate on the log grids the following sample code should work:

```
        Character*80 fname
        Character*24 key
        dimension temps(kt),rhos(krho),ross(kgray)
        .
        .
        .
        mat = matids(i)
        key = 'rgray'
        call ggeLgray(fname,mat,key,temps,kt,nt,rhos,krho,nrho,
 &              ross,kgray,ngray,ier)
        if(ier.ne.O)then
write(6,*)'Error from ggetgray, ier = ',ier
stop
        endif
        do it=l,nt
temps(it)=alog(temps(it))
        enddo
        do irho=l,nrho
 rhos(irho)=alog(rhos(irho))
        enddo
        do it=l,nt
do irho=l,nrhos
ross((it-l)*nrhos+irho))=alog(ross((it-l)*nrhos+irho))
enddo
        enddo
        .
        .
        .
        tlog=alog(tdesired)
        rlog=alog(rhodesired)
        call gintgrlog(temps,nt,rhos,nrho,ross,ngray,tlog,rlog,ans)
```

where ens is the interpolated Rosseland gray opacity. As an alternative the section taking logs of the data could be omitted and a call made to gintgray with the arguments the same as the call to gintgrlog.

There are similar subroutines for interpolating on multigroup quantities. The difference is that the multi-group data is passed to the program and an array is returned, representing the interpolated multi-group opacity with dimension nhnu-1, the number of photon groups. For more details see Appendix B.

V. Summary

We have described the subroutines for accessing data on the IPCRESS formatted files and interpolating on the opacity data. We have compiled a library of these routines called gandolflib.a. This load library along with source code for a test driver and a couple of sample IPCRESS data files are available on CFS for several computing platforms. All the subroutines and data files are unclassified. The test driver, script for compiling it, and data files are available as tar files for the Crays, SGI Origin 2000, and Sun platforms. They are stored on cfs as /oplib/machinename/gandmachinename.tar, where the string 'machinename' should be replaced with gamma for machine gamma, sun for a sun workstation, or asciblue for the bluemountain machine. These are available in both open and secure partitions.

Once the tar file has been retrieved on a platform, the command

```
tar xvf gandmachine.tar
```

Will extract all files. There will now be the following files in the local file space:

```
bliss
etafile
gandolflibmachinename.a
readme
tstgandolf.f
xtstmachinename
```

There is one difference when using the bluemountain machines. In that case there is a choice in using, or not using the -64 option in compiling. We have made load libraries with both options, so on the bluemountain machines there are two versions of gandolflibmachinename.a, gandolflib32.a and gandolflib64.a, respectively. There are also two script files, xtstblue32 and xtstblue64, to compile the test code, tstgandolf.f depending on whethor the -64 option is used or not.

To compile the test code, simply type in

```
xtstmachinename
```

And the code will be compiled with the executable named tstgandolf, except on the bluemountain machine where it will be named either tstgandolf32 or tstgandolf64.

There are two sample IPCRESS files, bliss and etafile. The bliss file is a standard multi-group opacity file containing gray and multi-group opacities, as well as other data, with associated temperature, density, and photon energy grids. The other file, etafile, also contains gray and multigroup data, but the density grid has been replaced with the electron degeneracy parameter grid, the eta grid. This is needed if it is desired to perform a mixture of multi-group opacities, rather than having the TOPS code perform the

mixture. In performing a mixture it is necessary that the constituents of the mixture all have the same electron pressure. This is insured by mixing at equal eta values, which is the reason for replacing the density grid with the eta grid for some files. The IPCRESS file format can store either type of table and the Gandolf subroutines can extract either set of data.

Once the tstgandolf code is compiled, it can be run. It will prompt for input and explain options.

Appendix A.

We list here the names of all the subprograms included in the Gandolf load library to help avoid name conflicts with user programs. Note that all the Gandolf subprograms begin with the letter g. We give the names of the Fortran subroutines first.

```
      subroutine gmatids(fname,matids,kmats,nmats,ier)
      subroutine gkeys(fname,mat,keys,kkeys,nkeys,ier)
      subroutine gchgrids(fname,mat,nt,nrho,nhnu,ngray,nmg,ier)
      subroutine ggeLgray(fname,mat,key,temps,kt,nt,rhos,krho,nrho,
     &                    data,kdata,ndata,ier)
      subroutine ggetmg(fname,mat,key,temps,kt,nt,rhos,krho,nrho,
     &     hnu,khnu,nhnu,data,kdata,ndata,ier)
      subroutine ggeedata(fname,mat,key,data,kdata,ndata,ier)
      subroutine gintgray(tgrid,nt,rgrid,nr,data,ndata,t,r,ansgr)
      subroutine gintgrlog(tgrid,nt,rgrid,nr,data,ndata,t,r,anagr)
      subroutine gintgrlin(tgrid,nt,rgrid,nr,data,ndata,t,r,ansgr)
      subroutine gintmg(tgrid,nt,rgrid,nr,nhnu,data,ndata,t,r,ansmg)
      subroutine gintmOlog(tgrid,nt,rgrid,nr,nhnu,data,ndata,t,r,ansmg)
      subroutine gintmOlin(tgrid,nt,rgrid,nr,nhnu,data,ndata,tir,ansmg)
      subroutine gchip(fname,ier)
      subroutine glJust(inname,outname,length)
      subroutine gninit(ifnam,riob,niobl,niob2,rnw,riad,rkey,
      subroutine gnsrch(riob,rnw,riad,rkey,nk,key,nw,iad)
      subroutine gr2disk(ifnam,rdata,numw,iad,ierr)
      subroutine grcdisk(ifnam,rdata,numw,iad,ierr)
      subroutine gfexist(file,iex)
```

We next list the names of the C functions. Note that these names will be modified for the correct type for calling from Fortran on different platforms. Thus, F_GCASSIGNR will become GCASSIGNR, GCASSIGNR_, gcassignr, or gcassignr_ depending on the platform.

```
      void F_GCASSIGNR(int *unit,char *fname,int *ierr)
      void F_GCPOSITION(unit,nLytes,ierr)
      void F_GCREAD(unit , array , ilen , ierr)
      void F_GCCLOSE(unit)
      void F_GCEXIST(char *fname,int *ierr)
```

Appendix B.

We list here the comment sections of the user callable subroutines. Included is the meaning of the value returned for ier from each subroutine.

subroutine gmatids(fname,matids,kmats,nmats,ier)

```
c
c       subroutine to get array of material ids from IPCRESS file
c
c
c       on input
c
c               fname  (character*80) name of IPCRESS file
c               kmats  (integer*8)    size of matids array in calling program
c
c
c       on output
c
c               matids (integer*8)    array of material ids found on file
c               nmats  (integer*8)    number of matids found
c               ier    (integer*8)    error number
c                       ier = 0  normal return
c                       ier = 1  requested file not found
c                       ier = 2  file is not IPCRESS
c                       ier = 3  problem in reading file
c                       ier = 4  no matids found
c                       ier = 5  too many matids for array
c
```

```
      subroutine gkeys(fname,mat,keys,kkeys,nkeys,ier)

c
c     subroutine to get keywords for a material on a file
c
c
c     on input
c
c           fname  (character*80)      name of IPCRESS file
c           mat    (integer*8)         material id to search
c           kkeys  (integer*8)         size of keys array in calling program
c
c
c     on output
c
c           keys   (character*24)      array of keywords found for mat
c           nkeys  (integer*8)         number of keys found
c           ier    (integer*8)         error number
c                  ier = 0  normal return
c                  ier = 1  requested file not found
c                  ier = 2  file is not IPCRESS
c                  ier = 3  problem in reading file
c                  ier = 4  no keys found for mat
c                  ier = 5  too many keys for array
c
```

```
      subroutine gchgrids(fname,mat,nt,nrho,nhnu,ngray,nmg,ier)
c
c     subroutine to get grid sizes for a material on a IPCRESS file
c
c
c     on input
c
c           fname (character*80) name of IPCRESS file
c           mat (integer*8)      material id to search
c
c
c     on output
c
c           nt      (integer*8)          number of temperatures for mat
c           nrho    (integer*8)    number of densities or etas for mat
c           nhnu    (integer*8)    number of photon energies for mat
c           ngray   (integer*8)    number of gray data for mat
c           nmg     (integer*8)    number of mg data for mat
c           ier     (integer*8)    error number
c                 ier = 0  normal return
c                 ier =-l  normal return with etas, not densities
c                 ier = l  requested file not found
c                 ier = 2  file is not IPCRESS
c                 ier = 3  problem in reading file
c                 ier = 4  inconsistent gray grids, mg not checked
c                 ier = 5  ngray not equal to nt*nrho, mg not checked
c                 ier = 6  inconsistent mg grids
c                 ier = 7  nmg not equal to nt*nrho*(nhnu-l)
```

```
      subroutine ggetgray(fname,mat,key,temps,kt,nt,rhos,krho,nrho,
     c     data,kdata,ndata,ier)
c
c     subroutine to get gray data for a material on a file
c
c
c     on input
c
c          fname (character*80) name of IPCRESS file
c          mat (integer*8)          material id to search
c          key (character*24)       keyword for gray data
c          kt (integer*8)  dimension of temps array
c          kr (integer*8)  dimension of rhos array
c          kdata (integer*8)   dimension of data array
c
c
c     on output
c
c          temps (real*8)      array of temperatures for mat
c          nt (integer*8)      number of temperatures for mat
c          rhos (real*8)       array of densities or etas for mat
c          nrho (integer*8)    number of densities or etas for mat
c          data (real*8)       gray data for keyword
c          ndata (integer*8)   number of gray data for mat
c          ier (integer*8)     error number
c               ier   = 0      normal return
c               ier   =-l      normal return with etas, not densities
c               ier   = l      requested file not found
c               ier   = 2      file is not IPCRESS
c               ier   = 3      problem in reading file
c               ier   = 4      data not found, check nt, nrho, ndata
c               ier   = 5      grid size exceeded, check all dimensions
c               ier   = 6      ndata not equal to nt*nrho
```

```
      subroutine ggetmg(fname,mat,key,temps,kt,nt,rhos,krho,nrho,
     c     hnus,khnu,nhnu,data,kdata,ndata,ier)
c
c     subroutine to get multi-group data for a material on a file
c
c
c     on input
c
c          fname  (character*80)        name of IPCRESS file
c          mat    (integer*8)           material id to search
c          key    (character*24) keyword for gray data
c          kt          (integer*8)           dimension of temps array
c          kr          (integer*8)           dimension of rhos array
c          khnu   (integer*8)           dimension of hnu array
c          kdata  (integer*8)           dimension of data array
c
c
c     on output
c
c          temps  (real*8)              array of temperatures for mat
c          nt          (integer*8)      number of temperatures for mat
c          rhos   (real*8)              array of densities or etas for mat
c          nrho   (integer*8)           number of densities or etas for mat
c          hnus   (real*8)              array of dphoton boundaries or mat
c          nhnu   (integer*8)           number of photon boundaries for mat
c          data   (real*8)              gray data for keyword
c          ndata  (integer*8)           number of gray data for mat
c          ier    (integer*8)           error number
c               ier    = 0    normal return
c               ier    =-l    normal return with etas, not densities
c               ier    = l    requested file not found
c               ier    = 2    file is not IPCRESS
c               ier    = 3    problem in reading file
c               ier    = 4    data not found check nt, nrho, nhnu, ndata
c               ier    = 5    grid size exceeded, check all dimensions
c               ier    = 6    ndata not equal to nt*nrho*(nhnu-l)
```

```
      subroutine gintgrlog(tgrid,nt,rgrid,nr,data,ndata,t,r,ansgr)
c
c     subroutine to interpolate gray data using linear on log
c
c     note - input values are assumed to be logs. output is actual value.
c
c     on input
c
c           tgrid   (real*8)      log of temperature grid
c           nt      (integer*8)   number of temperatures
c           rgrid   (real*8)      log of density grid
c           nr      (integer*8)   number of densities
c           data    (real*8)      log of gray data
c           ndata   (integer*8)   number of data points
c           t       (real*8)      log of requested temperature point
c           r       (real*8)      log of requested density point
c
c     on output
c
c           ansgr   (real*8)      interpolated data value at t,r
c                                 if t,r go off table, table boundary
c                                 value is used.

      subroutine gintmglog(tgrid,nt,rgrid,nr,nhnu,data,ndata,tirransmg)
c
c     subroutine to interpolate mg data using linear on log
c
c     note - input values are assumed to be logs. output is actual value.
c
c     on input
c
c           tgrid   (real*8)      log of temperature grid
c           nt      (integer*8)   number of temperatures
c           rgrid   (real*8)      log of density grid
c           nr      (integer*8)   number of densities
c           nhnu    (integer*8)   number of photon energies (ngroup=nhnu-l)
c           data    (real*8)      log of mg data
c           ndata   (integer*8)   number of data points
c           t       (real*8)      log of requested temperature point
c           r       (real*8)      log of requested density point
c
c     on output
c
c           ansmg(ngroup) (real*8)    interpolated data value at t,r
c                                     if t,r go off table, table boundary
c                                     value is used.
```

```
subroutine ggetdata(fname,mat,key,data,kdata,ndata,ier)

c
c       subroutine to get general data for a material on a file
c
c
c       on input
c
c               fname  (character*80) name of IPCRESS file
c               mat     (integer*8)   material id to search
c               key    (character*24) keyword for gray data
c               kdata  (integer*8)    dimension of data array
c
c
c       on output
c
c               data   (real*8)       gray data for keyword
c               ndata  (integer*8)    number of gray data for mat
c               ier    (integer*8)    error number
c                      ier  =  0      normal return
c                      ier  =  1      requested file not found
c                      ier  =  2      file is not IPCRESS
c                      ier  =  3      problem in reading file
c                      ier  =  4      data not found
```

Appendix C.

We present here a list of keywords with the description of the type of data represented.

Keyword	Description
tgrid	The temperature grid in keV
rgrid	The density grid in gm cm^{-2}
etagrid	The electron degeneracy grid
fl2eta	The Fermi l/2 integral of eta
hnugrid	The photon group boundaries in keV
ramg	The Rosseland multi-group absorption opacities
rsmg	The Rosseland multi-group scattering opacities
rtmg	The Rosseland multi-group total opacities
pmg	The Planck multi-group opacities
rgray	The Rosseland gray opacities
pgray	The Planck gray opacities
tfree	The mean ion charge
p	The total pressure
e	The total energy
pelect	The electron contribution to the pressure
eelect	The electron contribution to the energy
pnuc	The nucleus contribution to the pressure
enuc	The nucleus contribution to the energy
hnuplas	The plasma cutoff frequency in keV
comp	A summary of the composition for a mixture
general	Some general information about the IPCRESS file

Note that there is some equation of state (EOS) information on the opacity file. This is EOS information calculated by Group T-4 in the process of solving the LTE plasma model in order to arrive at the opacity cross section file. This EOS information should not be mistaken for the more extensive calculations performed by Group T-1 and maintained on the Sesame files. This EOS information is put on the files for illustrative purposes only.

Appendix D.

We now give a summary of the format for the IPCRESS file. The file is in three basic parts. First there is a prefix containing general file information. This prefix gives such information as the maximum number of keywords for an entry and locations of the other parts of the file. There is an index section that contains information on the individual data entries. This information is located through the keywords. A match in keywords will provide the number of words of data and disk address for that entry. Finally, there is the data section which is pointed to by the index section.

The prefix contains the following information:

Word 0 Must be either 'nirvana' or 'ipcress' as character string
Word l may be anything, it is not currently used.
Words 2-25 contain the values read into an index array with meaning as defined below.
In the following description we use the abbreviations da for disk address, ll for logical length, dd for disk length, nw for number of words of data for a data entry, and fad for the array of disk addresses for data entries. The variable lpre is the length of the prefix section which is 24. There is a provision for an intermediate directory section, the information block of length linf, but we are not currently using this section. Note that the first two words of the file are set aside as general identifiers so that the actual prefix information starts at address 2 (the first word has address of 0) .

riob(l)=da of nw array
riob(2)=da of fad array (riob(l)+mxrec)
riob(3)=logical length (ll) of prefix (24)
riob(4)=disk length (dl) of prefix
riob(5)=disk address (da) of prefix (2)
riob(6)=ll of information block (0)
riob(7)=dl of information block
riob(8)=da of information block
riob(9)=ll of index block
riob(10)=dl of index block (3*(mxkey+2)*mxrec)
riob(11)=da of keys (riob(2)+mxrec)
riob(12)=dl of data block used
riob(13)=dl of data block
riob(l4)=riob(10) + riob(11)
riob(15)=number of data records in data block
riob(16)=word length of key entry for data record
riob(17)=maximum number of search keys (mxkey)
riob(18)=da of last index entry (riob(l)+riob(15)-1)
riob(l9)=ll of last data record
riob(20)=dl of last data record
riob(21)=da of last data record
riob(22)=dl of file
riob(23)=last address on file

riob(24)=logical data space used

We make a few comments about this prefix. This prefix has been adopted from the older binary file format know as PARADISE. The PARADISE format had keywords that were combinations of floating point numbers, integers, and Hollerith. This made it difficult to use on different platforms. The new IPCRESS format eliminates the integers completely and uses character*24 values for the keywords. The PARADISE format kept the information on number of words and addresses of data entries together with the keywords in one array. In using this in a code the practice was to equivalence a floating point array with an integer array and use the appropriate array name to retrieve the integer value or floating point value of a keyword. We have tried to keep the file format as near the PARADISE format as possible, but there are now differences. Thus there are some entries in the riob array that do not make sense. They have been left in simply because they are not used in the current file handling and are sometimes used as checks on converting the PARADISE formatted files to IPCRESS.

We now look at some specific points. The maximum number of data entries that can be stored on an IPCRESS file is called mxrec. The value of mxrec is not explicitly stored on the file, but is obtained by the difference

mxrec = riob(2) - riob(l)

We note that there are some relationships among various elements of the riob array. In particular

riob(9) = (riob(17)+2) * riob(l5)
riob(l0) = (3*riob(16)+2) * riob(l7)
riab(l4) = riob(l0) + riab(11)

On the PARADISE file riob(l4) was the disk address of the first data record on the file. Due to splitting the keywords from the rest of the indexing and making the keywords character*24, we have lost that relationship, so currently riob(14) has no meaning. As noted above we are not currently using the information section so elements 6-7 of riob serve no purpose at this time. We make a distinction between logical length and disk length for entries. The difference is most apparent for the index section where disk space has been set aside (disk length), but only part has information stored on it (logical length). For the actual data entries the two should be identical. With the information given in the prefix it is a relatively simple matter to read in the nw, iad, and keys arrays. For a user supplied set of desired keys the index of the data entry is found. The nw array provides the number of words to be read and the fad array provides the disk address for the read. A simple random access read is then performed to read in the data record from the file.